Empirical analyses of the factors affecting confirmation bias and the effects of confirmation bias on software developer/tester performance

Conference or Workshop Item

Empirical Analyses of the Factors Affecting Confirmation Bias and the Effects of Confirmation Bias on Software Developer/Tester Performance

Gul Calikli
Software Research Laboratory
Department of Computer Engineering,
Bogazici University, Turkey
0090 212 3595400-7227
gul.calikli@boun.edu.tr

Ayse Bener
Ted Rogers School of Information Technology Management,
Ryerson University, Canada
0001 416 9795297
ayse.bener@ryerson.ca

ABSTRACT

Background: During all levels of software testing, the goal should be to fail the code. However, software developers and testers are more likely to choose positive tests rather than negative ones due to the phenomenon called confirmation bias. Confirmation bias is defined as the tendency of people to verify their hypotheses rather than refuting them. In the literature, there are theories about the possible effects of confirmation bias on software development and testing. Due to the tendency towards positive tests, most of the software defects remain undetected, which in turn leads to an increase in software defect density.

Aims: In this study, we analyze factors affecting confirmation bias in order to discover methods to circumvent confirmation bias. The factors, we investigate are experience in software development/testing and reasoning skills that can be gained through education. In addition, we analyze the effect of confirmation bias on software developer and tester performance.

Method: In order to measure and quantify confirmation bias levels of software developers/testers, we prepared pen-and-paper and interactive tests based on two tasks from cognitive psychology literature. These tests were conducted on the 36 employees of a large scale telecommunication company in Europe as well as 28 graduate computer engineering students of Bogazici University, resulting in a total of 64 subjects.

We evaluated the outcomes of these tests using the metrics we proposed in addition to some basic methods which we inherited from the cognitive psychology literature.

Results: Results showed that regardless of experience in software development/testing, abilities such as logical reasoning and strategic hypotheses testing are differentiating factors in low confirmation bias levels. Moreover, the results of the analysis to investigate the relationship between code defect density and confirmation bias levels of software developers and testers showed that there is a direct correlation between confirmation bias and defect proneness of the code.

Conclusions: Our findings show that having strong logical reasoning and hypothesis testing skills are differentiating factors in the software developer/tester performance in terms of defect rates. We recommend that companies should focus on improving logical reasoning and hypothesis testing skills of their employees by designing training programs. As future work, we plan to replicate this study in other software development companies. Moreover, we will use confirmation bias metrics in addition to product and process metrics in for software defect prediction. We believe that confirmation bias metrics would improve the prediction performance of learning based defect prediction models which we have been building over a decade.

Categories and Subject Descriptors

H.1.2 [**User/Machine Systems**]: Human Factors, Software Psychology

General Terms

Measurement, Experimentation, Human Factors

Keywords

Cognitive biases, confirmation bias, software engineering, software testing

1. INTRODUCTION

One of the basic components of software development and testing are the human aspects.

Among these human aspects are cognitive biases, which are defined as the deviation of human mind from the laws of logic and accuracy [1]. The notion of cognitive biases was first introduced by Tversky and Kahneman [2,3]. There are various cognitive bias types such as availability, representativeness, anchoring and adjustment.

PROMISE2010, Sep 12-13, 2010. Timisoara, Romania

As far as we know, Stacy and MacMillian are the two pioneers who recognized the possible effects of cognitive biases on software engineering [1]. Another study is by Parsons and Saunders [4], who empirically showed the existence of adjustment and anchoring on software artifact reuse.

Confirmation bias, which is one of these cognitive biases, is also likely to affect software development process, as it was previously indicated by Stacy and MacMillan [1]. The tendency of people to seek for evidence that could verify their theories rather than seeking for evidence that could falsify them is called *confirmation bias.* The term confirmation bias was first used by Peter Wason in his rule discovery experiment, where the subject must try to refute his/her hypotheses to arrive at a correct solution [5].

Wason also explained the results of his selection task experiment using facts based on confirmation bias [7]. In this task, Wason gave subjects partial information about a set of objects, and asked them to specify what further information they would need to tell whether or not a conditional rule ("If A, then B") applies. It has been found repeatedly that people perform badly on various forms of this test, in most cases ignoring information that could potentially *refute the rule*.

Empirical evidence shows that software testers are more likely to choose positive tests rather than negative tests [8]. However, during all levels of software testing the attempt should be to *fail the code* to reduce software defect density. In order to discover more defects, confirmation bias levels of testers and developers need to be low.

In this study, we propose a method to measure/quantify confirmation bias levels, so that empirical studies about the effect of confirmation bias on software development/testing can be carried out. Our methodology consists of interactive and written tests based on Wason's rule discovery and selection tasks, respectively. We analyze the outcomes of our tests based on the existing work in cognitive psychology literature as well as the metrics we have defined during this study.

The rest of the paper is organized as follows: Detailed information about confirmation bias and related work in cognitive psychology literature are given in Section II. We explain our methodology for measurement/quantification of confirmation bias in Section III. Metrics we defined for this study are explained in Section IV. We mention the dataset used in our empirical analysis in Section V. In Section VI results together with their corresponding interpretations are presented. Finally, the impact of the results and potential future directions are discussed in Section VII.

2. CONFIRMATION BIAS

This section explains the two experiments proposed by P. C. Wason [5,7] to show the presence of confirmation bias.

2.1 Wason's Rule Discovery Experiment

In this experiment, Wason asked his subjects to discover a simple rule about triples of numbers [5]. Initially, subjects are given a record sheet on which the triple "2, 4, 6" is written.

The experimental procedure can be explained as follows: The subjects are told that "2 4 6" conforms to this rule. In order to discover the rule, they are asked to write down triples together with the reasons of their choice on the record sheet. After each instance, the tester tells whether the instance conforms to the rule or not. The subject can announce the rule only when he/she is highly confident. If the subject cannot discover the rule, he/she can continue giving instances together with reasons for his/her choice. This procedure continues iteratively until either the subject discovers the rule or he/she wishes to give up. If the subject cannot discover the rule in 45 minutes, the experimenter aborts the test.

Wason designed this experiment in a way such that subjects mostly showed a tendency to focus on a set of triples that is contained inside the set of all triples conforming to the correct rule. Due to this fact, discovery of the true rule was possible only by *refuting* hypotheses that come to mind.

2.1.1 Eliminative/Enumerative Index

Wason's eliminative/enumerative index aims to give an idea about the kind of thinking of subjects by considering the nature of the instances given by the subjects in relation to their reasons for choice. This index is calculated as a ratio between the number of subsequent instances incompatible with each reason proposed to the number of compatible instances, summed over all proposed reasons. It is desirable to have eliminative/enumerative index to be greater than 1. Wason indicates that when this value is greater than 1 (the higher the better), the less confirmation bias of the subject is.

2.1.2 Test Severity

In [6], Poletiek mentions *severity* of the tests, which corresponds to the instances given by subjects, to discover the rule in Wason's selection task. A test is more severe when the chance of the supporting observation occurring under the assumption of the hypothesis *H* exceeds the chance of its occurring without the assumption of the *H* (i.e. with the assumption of the background knowledge *b* only). The higher this ratio is (exceeds 1), the higher the severity of the test is. In other words, when the severity of a test is high, more alternative hypotheses are eliminated.

2.2 Wason's Selection Task

In the original Wason's Selection Task, the subject is given four cards, where each card has a letter on one side and a number on the other side. These four cards are placed on a table showing respectively D, K, 3, 7. Given the hypothesis "Every card that has a D on one side has a 3 on the other side", the subject is asked which card(s) must be turned over to find out whether the hypothesis is true or false. The hypothesis can be translated into the logical implication of the form "If P, then Q", whereas each tests is the selection of one of the cards (P, not-P, Q, not-Q). Wason interprets selection of the cards D and 3 (i.e. P and Q) as a choice of a *verifier*, whereas the subject is defined to be a *falsifier* if he/she selects the cards D and 7 (i.e. P and not-Q). However, subject can choose cards D and 3 due to *matching bias* as well as *confirmation bias* [6, 9, 10].

2.2.1 Matching Bias

Matching bias may lead subjects to select cards on the basis of a simple judgment of relevance. In other words, the selection of the correct cards in the original Wason's selection task can also result due to matching of the letter D and number 3 in the stated

hypothesis. The separation of matching from logic requires use of rules of the form *if P, then Q* and three negated forms of the same rule, which are of the form *If P, then not Q, if not P, then Q* and *if not P, then not Q* respectively.

In [4], Evans and Lynch used the negated version of the selection task (i.e. if P, then not-Q) as well as the original task (i.e. if P, then Q). In this experimental study, the subjects chose P and Q cards, instead of P and not-Q cards. Evans and Lynch interpreted subjects' behavior as either being *falsifying* or *matching*. However, if a subject, who has chosen P and Q cards in the standard version, also selects P and Q cards in the negated version, such behavior can be explained only by matching bias. Otherwise, subject's verifying behavior accompanied by falsifying behavior would not make sense. In this study, all four negated forms are used to predict matching bias.

3. PROPOSED APPROACH TO MEASURE/ QUANTIFY CONFIRMATION BIAS

In order to conduct an empirical analysis, we need a methodology to measure/quantify confirmation bias level of individuals. For this purpose, we prepared two types of tests that are *interactive test* and *written test*.

3.1 Interactive Test

What we call *interactive test* is Wason's rule discovery task [5]. Interactive test was carried out just as the original task as mentioned before.

3.1.1 Calculation of Test Severity

There are various challenges in evaluating test severity. Firstly, the set of all possible hypotheses (i.e. background knowledge) is infinite. Secondly, humans cannot easily keep more than one hypothesis at a time [6]. On the other hand, according to Poletiek, a severe tester will not consciously formulate all hypotheses one by one, yet he/she will be able to make a globally accurate estimation [6]. Hence, it is not necessary to generate explicitly all possible alternatives in order to generate a more or less severe test.

In order to calculate test severity we followed the method employed by Poletiek in [6]. We took the set of hypotheses, generated by the subjects during our interactive tests, as the plausible set of hypotheses (i.e. background knowledge). For each instance given by the subject (i.e. test made by the subject), we followed the following procedure:

- If the test is *positive* (i.e. the instance given by the subject conforms to the rule to be discovered), then we took the number of hypotheses that are eliminated by the test as severity of the test. In other words, the hypotheses to which the given instance does not conform are taken into account.
- If the test is *negative* (i.e. the instance given by the subject does not conform to the rule to be discovered), then we took the number of hypotheses to which the given instance conforms, as severity of the test.

Table 1 shows the set of plausible hypotheses we generated using the rules announced by the subjects during our interactive tests. Our set of plausible alternatives consist of 27 hypotheses, hence severity of each instance given by a subject is within the range [0, 27].

3.1.2 Vincent Curves

As Wason defines in [5], Vincent curves represent performance of subjects towards a criterion, which is *not defined by fixed number of trials*. During interactive tests, total number of instances given before discovery of the correct rule varies from one subject to another. Hence, Vincent curves can be used to visualize the change in test severity of a group of subjects until the correct rule is discovered. Although, there are variants of Vincent curves, we use the original method proposed by Vincent as follows:

- Total number of instances given by each subject in the group is divided into *N* equal fractions.
- Within each fraction, we calculate the average of test severities of the instances that fall into that fraction. This calculation is done for each subject in the group.

For N equal fractions, N+1 data points are obtained per subject. The average of the i^{th} data point of all subjects gives the i^{th} data point for the group of subjects, where i = 1, 2,…, N+1.

We have selected total number of fractions (N) to be equal to the minimum number of instances given within the group before discovery of the correct rule. For number of instances which are not divisible by N, we used Vincent's original procedure. For instance, the division of 22 instances given by each subject among 5 fractions would be 5, 5, 4, 4, 4. In other words, 2 additional instances are distributed one by one, starting from the first fraction.

3.2 Written Test

Written test is based on Wason's selection task [7]. There are three different types of questions in the written test which are abstract questions, thematic questions and questions with software development theme.

Abstract questions require pure logical reasoning to be answered correctly; however some of this type of questions can also be answered correctly by matching. In our test, there are 8 abstract questions.

Thematic questions can be answered correctly using the cues produced by memory. This phenomenon where the stage of logical reasoning is bypassed is called *memory cueing* [8]. In our test there are 6 thematic questions which can be solved correctly through everyday life experience.

Questions with software development/testing theme are also thematic questions where pure logical reasoning can be bypassed by experience in software development and testing. Our test contains 8 questions of this type.

3.2.1 Determination of Existence of Matching Bias

Matching bias detection and classification of subjects as being falsifier, verifier or matcher can be done using abstract test results. In order to detect the existence of matching bias among subjects and classify them, we used all negated variants of Wason's original selection task.

- If there is a D on one side of the card, then there is a 3 on its other side

- If there is a D on one side of the card, then there is ***not*** a 3 on its other side
- If there is ***not*** a D on one side of the card, then there is a 3 on its other side
- If there ***not*** is a D on one side of the card, then there is ***not*** a 3 on its other side

Table 1. The plausible set of hypotheses used for test severity calculation

1	Integers ascending with increments of 2
2	Integers ascending with increments of k, where k = 1,2,...
3	Three integers in ascending order such that the average of the first and third integer is the second integer
4	The average of the first and third integer is the second integer
5	Even integers ascending with increments of 2
6	Integers ascending with increments of m = 2k, where k = 1,2,3, …
7	Integers ascending or descending with increments of m = 2k, where k = 1,2,3, …
8	Even integers in ascending order
9	Positive even integers in ascending order
10	Three even integers in any order
11	Three integers in any order, none of them are identical
12	Three integers in any order, two or three of them are identical
13	Three integers in ascending order such that difference between third and first number is even
14	Integers ascending or descending with increments of k, where k = 1, 2, 3, …
15	Sum of the first and second integer is the third integer
16	The triples of the form (2n 4n 6n), where n = 1,2,3, …
17	The triples of the form (n 2n 3n), where n = 1,2,3, …
18	Second integer is greater than the first one
19	Third integer is greater than the first integer
20	Difference between the third and the first integer is even
21	Greatest common divisor (GCD) of the integers is 2
22	Ascending integers such that each integer is 1 less than a prime number
23	Any three rational numbers
24	Positive real numbers in increasing order
25	Positive integers in increasing order
26	Three integers whose sum is even
27	Three even integers greater than zero

3.2.2 Falsifier/Verifier/Matcher Classification

As previously mentioned, given the conditional rule of the form *if P, then Q*, the subject who selects *P, Q* as the answer can either be a verifier or matcher. Similarly, the same answer for the rule *if P, then not-Q*, means that the subject can be a falsifier or a matcher. In order, to overcome this fuzziness, we employ the method of Reich and Ruth [14], which is explained below as follows:

- choice of not-Q in the rule "If P, then Q" = *falsifying*
- choice of not-Q in the rule "If P, then not Q" = *verifying*
- choice of P in the rule "If not P, then Q" = *matching*
- choice of not-Q in the rule "If not-P, then Q" = *falsifying*
- choice of P in the rule "If not P, then not Q" = *matching*
- choice of not-Q in the rule "If not P, then not Q" = *verifying*

This method of determining response tendencies is advantageous, as it does not confound strategies that might have contributed to a particular selection. However, it neglects a large proportion of data provided by the subjects. On the other hand, it gives a general view about the subjects' responses and it is the only classification strategy we came across in the existing psychology literature. For these reasons, we used the method of Reich and Ruth and we labeled subjects, whom we could not classify, as *None*.

4. METRICS

In order to perform empirical analysis, we also defined some metrics, in addition to the metrics and methodologies we inherited from cognitive psychology literature. Other than Wason's eliminative/enumerative index ($Ind_{Elim/Enum}$), the remaining metrics have been defined by us.

Among interactive test metrics, total time it takes to discover the correct rule (T_I) and total number of rule discovery attempts (N_A) are performance metrics. On the other hand, frequency of immediate rule announcements (F^{IR}), average length of consecutive immediate rule announcements (avg_L^{IR}) and average frequency of reason repetition/reformulation (avg_F^{RR}) are supposed to measure the extend of experimental procedure violation. The experimental procedure does not allow immediate rule announcements. However, during interactive tests some subjects made immediate rule announcements, although they had been told the experimental procedure at the beginning.

Written test metrics measure performance in different sections of the written test. These are the score in abstract questions (S_{ABS}), thematic questions (S_{Th}) and questions with software development/testing theme (S_{SW}) respectively. Each score metric is calculated as the ratio of the number of correctly answered questions to the total number of abstract questions. In addition, total duration it takes to solve thematic and abstract sections (T_{Th+ABS}) and the duration it takes to solve the sections with software development/testing theme (T_{SW}) are among written test metrics. All of the metrics are given in Table 2 together with their explanations.

5. DATA

We conducted both interactive and written tests to two different groups of subjects.

The first group (Group 1) consists of 28 computer engineering graduate students of Bogazici University. 14 of the subjects in Group 1 have software development experience in various companies for more than 2.51 years on average. Among subjects having software development experience above 2.51 years, 6 of them are still active and they are developing embedded software for RoboCup, which is an international robotics competition founded in 1993.

Members of Group 2 are software developers/testers working in a large scale telecommunication company in Europe. Unlike subjects of Group 1, this group of subjects has *only* undergraduate degrees in Computer Engineering, Mathematics and related fields. There are two different project groups within Group 2. The first project group, which employs traditional waterfall software development methodology, consists of 28 subjects. Among these 28 subjects, 12 of them are developers, while 16 of them are testers. The second project group consists of 8 subjects who develop software using TSP/PSP methodology.

Table 2. Interactive and written test metrics with their abbreviations

Interactive Test Metrics	
Abbr.[1]	**Metric Explanation**
$Ind_{Elim/Enum}$	Wason's eliminative/enumerative index [5]
T_I	Total time it took to discover the correct rule
F^{RR}	Immediate rule announcement frequency
avg_L ^{IR}	Total number of rule announcements in a series, where no instances are given in between rule announcements
avg_F^{RR}	Average frequencies of reason repetition/ reformulation
N_A	Total number of rule discovery attempts including the correct rule announcement
Written Test Metrics	
Abbr.	**Metric Explanation**
S_{ABS}	Score in abstract questions
S_{Th}	Score in thematic questions
T_{Th+ABS}	Duration it took to solve abstract and thematic questions (minutes)
S_{SW}	Score in questions with software development/testing theme
T_{SW}	Duration it took to solve questions with software development/testing theme (minutes)

[1] Abbr. stands for "Abbreviation".

6. RESULTS

This section consists of two parts. In the first part, the effects of factors such as education, experience in software development/testing and software development methodologies, on confirmation bias are analyzed. In the second part, we investigate the effects of confirmation bias on software development and testing.

6.1 Analysis of the Factors Affecting Confirmation Bias:

6.1.1 Effect of Education on Confirmation Bias

As shown in Figure 1, according to Reich and Ruth's classification method, there are more *falsifiers* and less *verifiers* in Group 1, compared to Group 2. These results imply that subjects of Group 1 exhibit lower confirmation bias levels. In addition, existence of *matchers* only in Group 2 (13.16% of Group 2 population) supports the fact that members of Group 1 use more logical reasoning. These results are in favor of Group 1 members, who are graduate computer engineering students and obliged to take theoretical computer science courses according to the graduate curriculum. It is highly probable that these courses helped Group 1 members to gain skills to perform logical reasoning, since they frequently experienced the fact that a given statement does not always have to be true and hence it may require to be disproved. In other words, Group 1 members have been trained to lower their confirmation bias levels through courses that require logical reasoning.

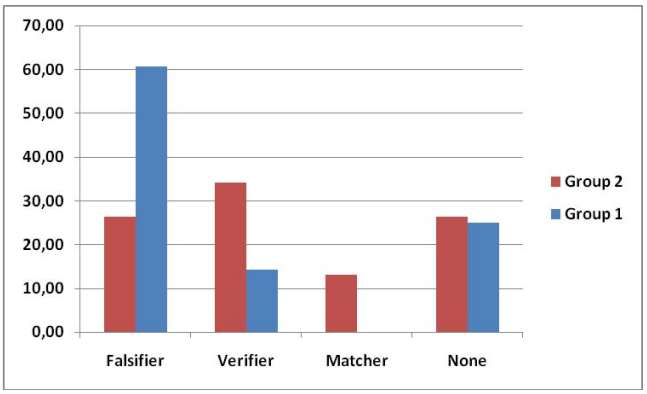

Figure 1. Distribution of falsifiers, verifiers, and matchers in Group 1 and Group 2 according to Reich and Ruth's method.

6.1.2 Effect of Software Development/Testing Experience on Confirmation Bias

In order to see how confirmation bias levels are affected by experience in software development/testing, we performed three different analyses. In our first analysis, we compared interactive and written test metric values of two subgroups within Group 1. The first subgroup (Group1_EXP) consists of subjects who have worked in software development industry for more than or equal to 2.51 years, which is the average years of experience among Group 1 members. The rest of the subjects are categorized under the second subgroup (Group1_NEXP). In order to compare interactive and written test metric values of Group1_EXP and

Group1_NEXP, we performed bootstrapped Kolmogorov-Smirnov test. As shown in Table 3, the only significant difference obtained is in the scores of the written test with software development and testing theme (S_{SW}). Members of group who have experience in software development/testing scored significantly higher, since in written test they used their software development knowledge gained through experience, in addition to logical reasoning.

In the second analysis, we employed the Reich and Ruth categorization method. The distribution of falsifiers and verifiers, as well as those that could not be categorized is shown in Figure 2. 64.29% of experienced members in Group 1 and 57.29% of members of Group 1 with less experience are falsifiers. However, 21.43 % of experienced Group 1 members and 7.14% of less experienced members are verifiers. These distribution results imply no significant difference among experienced and less experienced members of Group 1.

In the third analysis, we statistically compared experienced members of Group 2 ($Group2_{EXP}$) and less experienced Group 2 ($Group2_{NEXP}$). As shown in Table 4, no significant difference was found among the members of $Group2_{EXP}$ and $Group2_{NEXP}$. $Group2_{NEXP}$ consist of Group 2 members who have less than 5.71 years which is the average years of experience in software development/testing among Group 2 members

Table 3. Results of the bootstrapped Kolmogorov-Smirnov test among experienced and less experienced members of Group 1.

	$Group\ 1_{EXP}$	$Group\ 1_{NEXP}$	p-value
$Ind_{Elin/Enum}$	1.3029	0.6538	0.3775
T_I	8.6429	6.6923	0.1310
F^{IR}	0.1429	0.6124	0.1150
avg_L^{IR}	0.1429	0.2692	0.2205
avg_F^{RR}	0.6786	0.9746	0.5455
N_A	1.7857	2.6154	0.5365
S_{ABS}	0.5914	0.6350	0.7195
S_{Th}	0.8823	0.9064	0.5405
T_{Th+ABS}	15.0714	12.7857	0.2740
S_{SW}	0.8308	0.7186	0.0010
T_{SW}	11.1429	12.3571	0.2865

6.1.2.1 Effect of Activeness in Software Development/Testing

In addition to experience, the effect of activeness in software development/testing, on confirmation bias needs to be explored. For this purpose, we divided experienced members of Group1 ($Group1_{EXP}$) into two subgroups, namely $Group1_{ACTIVE}$ and $Group1_{INACTIVE}$. $Group1_{ACTIVE}$ consists of computer engineering graduate students who has experience in software development//testing and who are still developing/testing software. The members of this group develop embedded software for autonomous robots. The rest of the $Group1_{EXP}$ members are not active in software development/testing anymore and they are

Table 4. Results of the bootstrapped Kolmogorov-Smirnov test among experienced and less experienced members of Group 2.

	$Group\ 2_{EXP}$	$Group\ 2_{NEXP}$	p-value
$Ind_{Elin/Enum}$	1.11	1.12	0.6899
T_I	18.06	16.59	0.3874
F^{IR}	1.00	0.67	1.0000
avg_L^{IR}	0.55	0.53	1.0000
avg_F^{RR}	1.17	0.80	0.8644
N_A	3.61	2.18	0.1170
S_{ABS}	0.19	0.13	0.3874
S_{Th}	0.72	0.71	0.9313
T_{Th+ABS}	18.12	14.5	0.2336
S_{SW}	0.46	0.53	0.9303
T_{SW}	17.59	14.41	0.3874

mostly engaged in research studies. Table 7 shows the statistical comparison of the metric values for $Group1_{ACTIVE}$ and $Group1_{INACTIVE}$. As it can be seen, no significant difference has been observed in metric values within the 0.05 significance level.

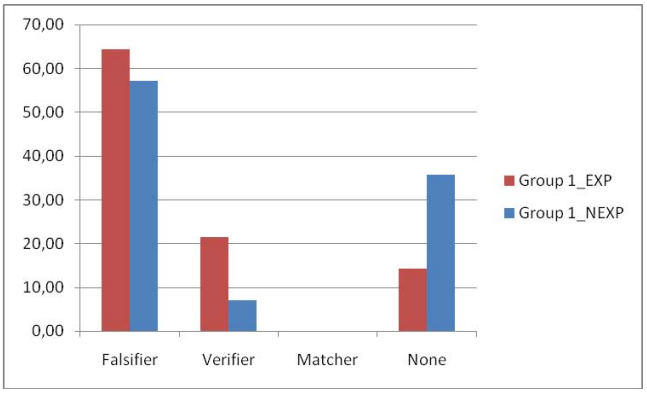

Figure 2. Distribution of falsifiers, verifiers, and matchers among the experienced and less experienced members of Group 1 according to Reich and Ruth's method.

We have also categorized members of $Group1_{ACTIVE}$ and $Group1_{INACTIVE}$ separately as *falsifiers*, *verifiers* and *matchers* according to Reich and Ruth's method. In both subgroups, subjects that could not be categorized according to the Reich and Ruth's scheme are labeled as *None*. As previously mentioned and shown in Figure 1, no *matchers* were found among the members of Group 1. Hence, we cannot observe any matchers in Figure 2 either. However, results seem in favor of $Group1_{INACTIVE}$ members, as a higher portion of $Group1_{INACTIVE}$ population is *falsifiers* and a lower portion of the population is *verifiers* when

compared to the falsifier and verifier portions within the $Group1_{ACTIVE}$ population.

When we consider Figure 1 and Figure 3 together, we can make the following observation: Among groups of subjects that consist of members active in software development/testing, lower portion of falsifiers and higher portion of verifiers are observed. This is an undesired situation as it implies high confirmation bias levels. In order to further investigate this claim of ours we conducted the following analysis: We removed 6 members who are still active in software development/testing from Group 1. We named the resulting group as Group 1'. We used Reich and Ruth's categorization method on the members of this group and compared the distribution of falsifiers, verifiers and matchers within Group 1' with the one in Group 2. Figure 4 shows the resulting categorization scheme, where higher portion of Group 1' population is falsifier; whereas verifiers form a lower portion, compared to the falsifier and verifier portions of Group 2.

During our analysis, we took into account only 12 developers of Group1 who develop software based on waterfall methodology and named this subgroup as $Group2_{REGULAR}$. As shown in Table 6, no significant statistical difference. As we can see in Figure 5, according to Reich and Ruth classification scheme, a higher portion falsifiers and a lower portion of verifiers are observed in $Group2_{REGULAR}$ compared to falsifier and verifier portions in $Group2_{TSP/PSP}$ population. These results seem in favor of $Group2_{REGULAR}$. However, 8.33% of $Group2_{REGULAR}$ are *matchers*, who cannot excel logical reasoning. Moreover, in both subgroups $Group2_{REGULAR}$ and $Group2_{TSP/PSP}$, a high portion of uncategorized subjects are observed.

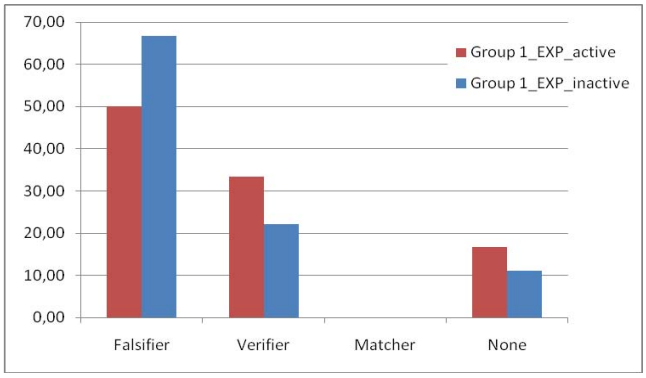

Figure 3. Distribution of falsifiers, verifiers, and matchers among the experienced active and experienced inactive members of Group 1 according to Reich and Ruth's method.

6.1.3 Effect Waterfall and TSP/PSP Software Development Methodologies on Confirmation Bias

In order to analyze the effect of waterfall and TSP/PSP software development methodologies, we statistically compared the interactive and written test metric values for two subgroups within group 2. As mentioned previously, 28 members of Group 1 are software developers/tester assigned to a software development projects using the regular waterfall methodology. Remaining 8 members of Group 2 ($Group2_{TSP/PSP}$) are responsible from a pilot software development project following TSP/PSP methodology. Among members of TSP/PSP group, 3 of them gave up the interactive test before discovering the correct rule. The interactive test metrics T_I and N_A can be measured only when a subject succeeds to discover the correct rule. Only 5 values for each metric exist, which is unlikely to give accurate results. Hence, during statistical comparison of metric values among these two groups, T_I and N_A metrics have been excluded.

Table 5. Results of the bootstrapped Kolmogorov-Smirnov test among experienced members of Group 1, that are active in software development/testing and those that are not.

	$Group1_{ACTIVE}$	$Group1_{INACTIVE}$	p-value
$Ind_{Elin/Enum}$	0.9160	1.5178	0.4505
T_I	10.4000	7.6667	0.7160
F^{IR}	0.0000	0.2222	0.0505
avg_L^{IR}	0.0000	0.2222	0.0515
avg_F^{RR}	1.1000	0.4444	0.3890
N_A	1.2000	2.1111	0.6920
S_{ABS}	0.5870	0.5300	0.5780
S_{Th}	0.8600	0.8667	0.3595
T_{Th+ABS}	16.6667	14.7778	0.5870
S_{SW}	0.8417	0.7689	0.3350
T_{SW}	12.6667	10.1111	0.4910

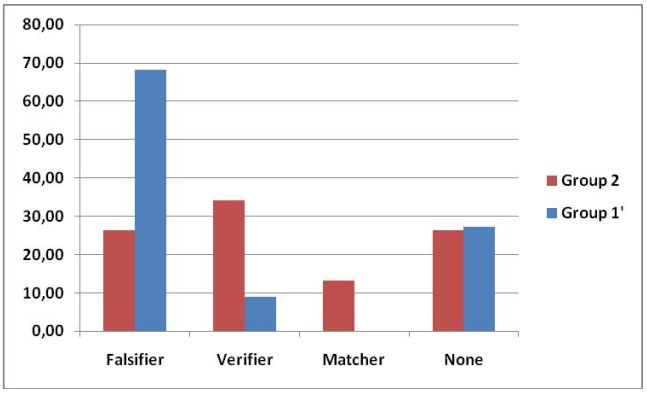

Figure 4. Distribution of falsifiers, verifiers, and matchers among members of Group 1 and Group 2 according to Reich and Ruth's method.

During our analysis, we took into account only 12 developers of Group1 who develop software based on waterfall methodology and named this subgroup as $Group2_{REGULAR}$. As shown in Table 6, no significant statistical difference. As we can see in Figure 5, according to Reich and Ruth classification scheme, a higher portion falsifiers and a lower portion of verifiers are observed in $Group2_{REGULAR}$ compared to falsifier and verifier portions in $Group2_{TSP/PSP}$ population. These results seem in favor of $Group2_{REGULAR}$. However, 8.33% of $Group2_{REGULAR}$ are *matchers*, who cannot excel logical reasoning. Moreover, in both subgroups

Group2$_{REGULAR}$ and Group2$_{TSP/PSP}$, a high portion of uncategorized subjects are observed.

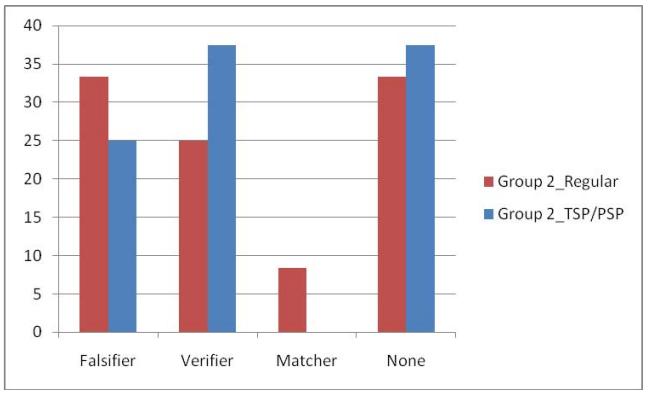

Figure 5. Distribution of falsifiers, verifiers, and matchers among members of Group2$_{REGULAR}$ and Group2$_{TSP/PSP}$ according to Reich and Ruth's method.

Table 6. Results of the bootstrapped Kolmogorov-Smirnov test among members of Group2$_{REGULAR}$, and Group2$_{TSP/PSP}$.

	Group2$_{REGULAR}$	Group2$_{TSP/PSP}$	p-value
$Ind_{Elin/Enum}$	1.1192	1.0938	0.5630
F^{IR}	1.1667	0.5000	0.2865
avg_L^{IR}	0.6250	0.5000	0.2930
avg_F^{RR}	1.1458	0.7500	0.3480
S_{ABS}	0.2942	0.3287	0.4875
S_{Th}	0.8192	0.7913	0.3995
T_{Th+ABS}	16.5000	14.1250	0.5765
S_{SW}	0.6483	0.5800	0.4990
T_{SW}	16.5833	12.5000	0.5325

6.2 Analysis of the Effects of Confirmation Bias on Software Development and Testing Performances

6.2.1 Effect of Confirmation Bias on Software Development Performance

We performed an analysis among 28 members of Group 1, who are all belong to a project group responsible from the development of the customer services software package. Within this project group, which develops software according to the traditional waterfall methodology, software testing team consists of 11 software testers, while the remaining 17 subjects are software developers. Every two weeks, a new release of the software is delivered and hence testing phase of one release and the development phase of the next release overlap. In this study, we analyzed 10 releases of the software that were developed and tested between the last week of May 2009 and second week of November 2009. For each release, we categorized each file to be defected or not based on the results of the testing phase for that release. Moreover, a file that was updated or created within a specific release but not updated during the following releases, was also categorized as defective if defects were found in that file during the testing phase of the following releases. For defects detected within a file during each testing phase, developers who created and updated that file before that testing phase were held responsible.

Based on the commit history of the files comprising the software package, we discovered that most of the files were updated by more than one developer. In other words, each file is developed by a group of one or more developers. As a result of churn data analysis, we found 124 developer groups and for each developer group we evaluated the defected file percentage among all the files created or updated by that group. Defected file percentage of each group is the measure we have selected to assess performance of each group of software developers. For each developer group, we also evaluated the average, minimum and maximum values of the 11 confirmation bias metrics that were listed in Table 4. In addition to confirmation bias metrics, we took into account the average, minimum and maximum test severity values to assess the hypotheses testing performance of a subject during the interactive test. Our method to evaluate the confirmation bias related parameters can be formulated as follows:

$$X_1 = \overline{Ind}_{E\lim/Enum}^{1...N} \qquad X_2 = Ind_{E\lim/Enum}^{1...N}$$

$$X_3 = T_I^{1...N} \qquad X_4 = \left(F^{IR}\right)^{1...N}$$

$$X_5 = \left(avg\_L^{IR}\right)^{1...N} \qquad X_6 = \left(avg\_F^{RR}\right)^{1...N}$$

$$X_9 = S_{Th}^{1...N} \qquad X_{10} = T_{Th+ABS}^{1...N}$$

$$X_{11} = S_{SW}^{1...N} \qquad X_{12} = T_{SW}^{1...N}$$

$$X_{13} = TestSeverity_{avg}^{1...N}$$

$$X_{14} = TestSeverity_{\min}^{1...N}$$

$$X_{15} = TestSeverity_{\max}^{1...N} \qquad (1)$$

$$X_{avg} = \left[\frac{\sum_{i=1}^{N} X_1^i}{N} \quad \cdots \quad \frac{\sum_{i=1}^{N} X_{15}^i}{N} \right] \qquad (2)$$

Each X_2-X_{12} are the confirmation bias metrics given in Table 2, while X_1 is elimination/enumeration index taking into account only the last rule announcement instead of every rule announcement made by the subject. Finally, X_{13}, X_{14} and X_{15} are respectively average, minimum and maximum test severities of each developer in a given group.

Having evaluated confirmation bias related parameters of

developer groups, we performed multi-linear regression modeling to find the relation between confirmation bias and percentage of defected files.

$$y = X\beta + \varepsilon \quad (3)$$

Table 7. The values of regression coefficients with their confidence intervals.

Coefficient	Coefficient Value	Confidence Interval	p value
β_1	6.5669	6.0569 - 7.0688	1.0791E-12
β_2	0.2696	0.0507 - 0.4896	0.0162
β_3	-0.1472	-0.4809 - 1.1866	0.3843
β_4	1.4814	1.0971 - 1.8657	6.543E-12
β_5	0.6248	0.0496 - 1.2000	0.0335
β_6	-1.2697	-1.9005 - -0.6309	1.167E-4

Since the existence of linear dependency leads to matrix singularity problem, we performed principal component analysis (PCA). Hence, we constructed a multiple linear regression model with 5 parameters (i.e. β_2 , β_3 , β_4 , β_5 , β_6) which are the linear combinations of average confirmation bias related parameters (i.e. $X = X_{avg}$) The coefficients for the resulting parameters that turned out to significantly contribute to the model together with their confidence intervals at $\alpha = 0.05$ significance level, are shown in Table 7. The R^2 statistic is 0.4477 and the adjusted R^2 statistic is 0.4243 which implies that about 42% of the variability in defect percentage is explained by the parameters given in Table 7. If we take into account the fact that defect rate is affected by process and many human attributes other than confirmation bias the results obtained are quite significant.

6.3 Analysis of the Effects of Confirmation Bias on Tester Performance

In this part of our work, we analyzed the effect of confirmation bias on tester performance. For this purpose, we inherited two tester performance metrics from tester competence reports of the company among whose employees are members of Group 2. These metrics are the number of bugs reported (N_{BUG}) and the number of production defects caused (N_{PROD_DEF}) by each tester respectively. Production defects are the defects that could not be detected by testers during testing phase and they are revealed by customers after the software is released. We grouped members of Group 2 based on the values of N_{BUG} and N_{PROD_DEF}.

Figure 6 shows the Vincent Curves for test severity values of two groups of testers. Testers are grouped according to the number of bugs reported by them as testers reporting above and below average number of bugs. On the contrary to what we have expected, group of testers reporting bugs *below* average value had exhibited a more strategic approach during interactive confirmation bias tests. This group of testers starts with a low level severe test and they progressively exclude more alternatives [6]. Moreover, starting from the second percent of the instances given during the interactive tests, test severity of the tester group having N_{BUG} value lower than average is always higher than that of the other group. In other words, for each instance given by members of this group during the interactive test more alternative hypotheses are eliminated. We can make an analogy between the testing strategies exhibited by the members of this group during interactive confirmation bias tests. The testers with N_{BUG} value below average seem to run tests that eliminate more software failure scenarios during the software testing phase. However, such a behavior is an expected result in finding more of the bugs in the code.

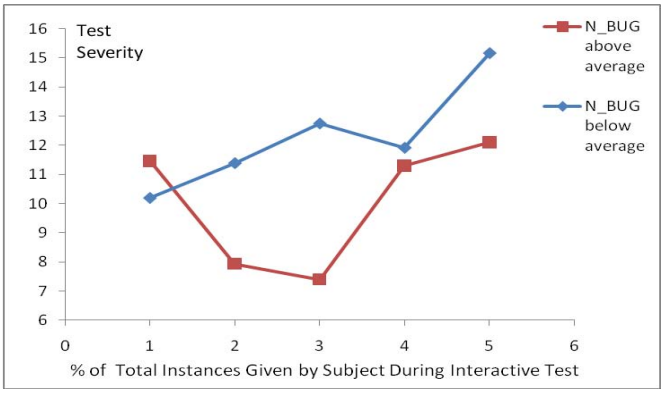

Figure 6. Vincent curves for test severity of testers who report bugs above and below average respectively.

In order to explain this, we analyzed the relationship between total number of bugs reported (N_{BUG}) and total number of production defects caused by each tester (N_{PROD_DEF}). The Spearman correlation value between these two variables is ***0.8234***, where +1 or -1 occurs when each of the variables is a perfect monotone function of the other. As shown in Figure 8, while the total number of bugs reported by a tester increases, total number of production defects introduced by that tester also increases.

High correlation between total number of reported bugs and production defect count may indicate another phenomenon, namely, testers who report more bugs might be assigned codes with very high defect density requiring immense testing effort. However, for each tester there is also a time pressure to end the testing procedure and this may result in the deployment of the defected codes. Another explanation for the outcome shown in Figure 8, is that bugs are not classified according to their severities. Hence, large number of reported bugs does not necessarily mean that a significant portion of severe bugs has been reported.

Moreover, as shown in Figure 10 testers who report bugs more than the average number of reported bugs ($N_{BUG\_above\ average}$) are less likely to follow a testing strategy in terms of test severity during interactive tests. A reasonable testing strategy suggested by Poletiek is to start with a low level severe test and to progressively increase test severity [6]. The test severity curve of testers who report bugs less than the average, is in line with Poletiek's testing strategy compared to the curve of the testers who report bugs below average. In addition, when the percentage of total instances given by subjects during the interactive test exceeds 10%, the test severity of testers who report bugs below average is always higher.

This outcome of interactive test suggests that the testers are more likely to follow Poletiek's testing strategy during software testing,

so that initially less severe tests are made. Hence fewer bugs are detected, yet tester gains an idea about the sections of the code that must be tested and possible defect types. As a result, tester can increase the severity of his/her tests which leads his/her finding more bugs that are severe.

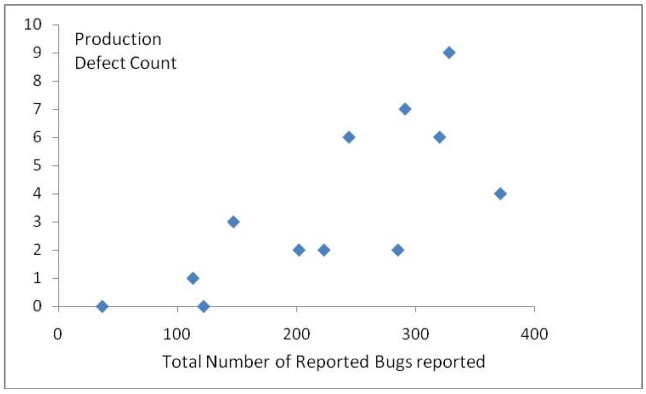

Figure 8. High correlation between production defect and total number of reported bugs (Spearman rank correlation: 0.8234)

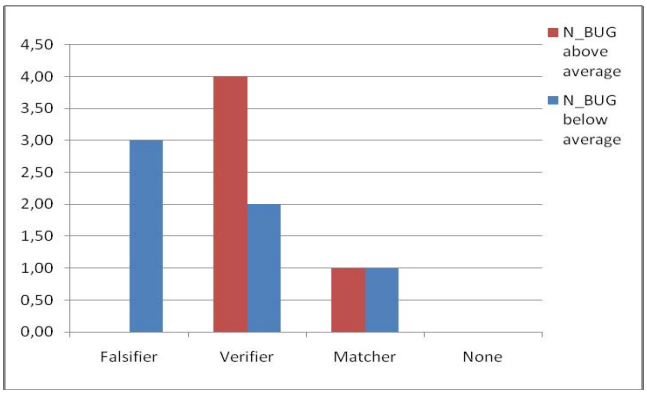

Figure 9. Distribution of falsifiers, verifiers, and matchers among testers who report bugs above and below average amount, according to Reich and Ruth's method.

Finally, as shown in Figure 9, all falsifiers are among testers who report bugs below average, whereas a higher portion of the testers who report bugs above average are verifiers. This result brings about the possibility that testers who report bugs above average exhibit more tendency to verify that production defects do not exist in the codes they test. Therefore, they exhibit confirmation bias in this sense.

The distribution of falsifiers, verifiers and matchers for testers who cause production defects above and below average is also in line with the distribution given in Figure 11. In addition, as shown in Figure 10, test severity curves for testers who cause production defects below and above average exhibit a behavior similar to the curves in Figure 6.

6.4 Threats to Validity

We would like to address internal, external, construct, and statistical validity.

In terms of internal validity, our quasi-independent variables are experience, education, activeness in software development, and software development methodology. The measures for these variables, which are confirmation bias metrics were taken within a week for both Group 1 and Group 2. Moreover, within any of the groups there was no event in between the confirmation bias tests that can affect subjects' performance.

However, problem may arise due to different experimental conditions. For instance, compared to graduate computer engineering students, stress factor of company workers due to the fact that they always have to rush the next release may have biased the results. In order to avoid mono-operation bias as a construct validity threat, we used more than a single dependent variable. We extracted metrics from both written and interactive tests as well as Wason's elimination/enumeration index [5]. As a result, we have avoided under-representing the construct and got rid of irrelevancies.

We have used two datasets to externally validate our results. We will continue expanding the size and variety of our dataset going forward.

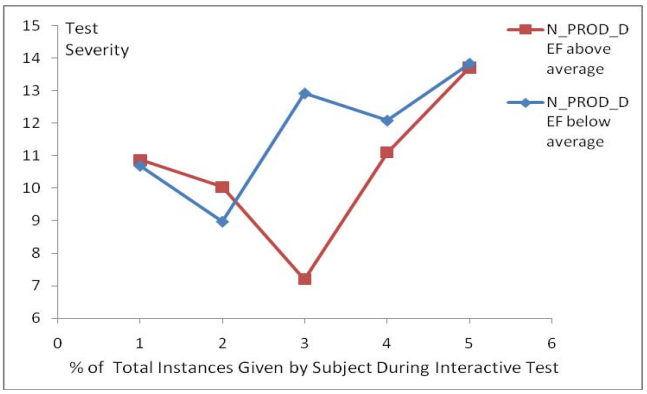

Figure 10. Vincent curves for test severity of testers who cause production defect above and below average respectively.

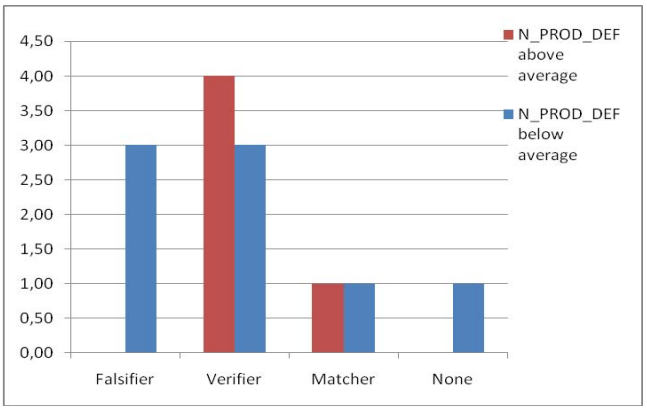

Figure 11. Distribution of falsifiers, verifiers, and matchers among testers who cause production defects above and below average amount, according to Reich and Ruth's method.

We used bootstrapped Kolmogorov-Smirnov tests to statistically validate our results. We used this test since we do not have any prior knowledge of the distribution of the metric values and the underlying distributions are discontinuous.

7. CONCLUSION AND FUTURE WORK

During all levels of software testing the attempt should be to *fail the code* to reduce software defect density. In an early work, Teasley et al. empirically showed that people have more tendency to make positive tests rather than negative tests during software testing phase due to confirmation bias [8].

In order to empirically analyze the effect of confirmation bias on software defect density, we need to measure/quantify confirmation bias. In this study, we prepared both interactive and written tests based on Wason's experiments that have been replicated for decades. However, unlike other disciplines, to the best of our knowledge, Wason's experiments have not been used in the field of software testing and development. Having performed our tests to testers and developers of a large scale telecommunication company in Europe as well as a group of computer engineering graduate students, we analyzed these test results based on the existing work in the cognitive psychology literature as well as the metrics we defined. Our results can be summarized as follows:

- Confirmation bias levels of individuals who have been trained in logical reasoning and mathematical proof techniques are significantly lower. In other words, given a statement such individuals show tendency to refute that statement rather than immediately accepting its correctness.
- A significant effect of experience in software development/testing has not been observed. This implies that training in organizations is focused on tasks rather than personal skills. Considering that the percentage of people with low confirmation bias is very low in the population [5, 6, 7], an organization should find ways to improve basic logical reasoning and strategic hypothesis testing skills of their software developers/testers.
- Individuals, who are experienced but inactive in software development/testing, score better in confirmation bias tests than active experienced software developers/testers. This implies that companies should balance work schedule of testers similar to jet pilots and allow them periodically to take some time off the regular routine.
- Another finding is that we do not observe any difference in confirmation bias levels in favor of the TSP/PSP team. This raises a question on the validity of models such as TSP/PSP that are promising defect free and high quality software development.
- High levels of defect rates introduced by software developers are directly related to confirmation bias.
- High levels of confirmation bias among software testers are very likely to result in an increase in the number of production defects.

As future work, we plan to extend our dataset and replicate this study in other software development companies. Moreover we will construct software defect prediction models that use confirmation bias metrics as people related set of metrics in addition to product and process metrics. It is highly probable that confirmation bias metrics would improve the prediction performance of learning based defect prediction models which we have been building over a decade.

8. ACKNOWLEDGMENTS

This research is supported in part by Turkish Scientific Research Council, TUBITAK, under grant number EEEAG108E014.

8. REFERENCES

[1] Stacy, W. and MacMillan, J., 1993. Cognitive bias in software engineering. Communication of the ACM. 38, 6 (June 1995), 57-63. DOI= http://doi.acm.org/10.1145/203241.203256

[2] Kahneman D., Slovic P., and Tversky, A. (Eds.) 1982 *Judgment Under Uncertainty: Heuristics and Biases*. New York: Cambridge University Press ISBN 978-0521284141

[3] Tversky, A. and Kahneman, D. 1971. Belief in the law of small numbers. Psychological Bulletin, 76, 105-110.

[4] Parsons, J. and Saunders, C., IEEE Transactions on Software Engineering. 30, 12 (December 2004), 873-888.

[5] Wason, P. C. 1960. On the failure to eliminate hypotheses in a conceptual task. Quarterly Journal of Experimental Psychology (Psychology Press), 12. 129–140.

[6] Poletiek, F. 2001 Hypothesis Testing Behavior (Essays in Cognitive Psychology). Psychology Press Ltd.

[7] Wason, P. C. 1968. Reasoning about a rule. Quarterly Journal of Experimental Psychology (Psychology Press) 20: 273–28.

[8] Teasley, B., Leventhal, L. M., and Rohlman, S. Positive test bias in software engineering professionals: What is right and what's wrong. In *Empirical Studies of Programmers: Fifth Workshop*, C.R. Cook, J.C. Scholtz, and J. C. Spohrer, Eds.1993.

[9] Evans, J. St. B. T., Newstead, S. E. and Byrne, R. M. 1993 *Human Reasoning: The Psychology of Deduction.*East Sussex, U.K.: Lawrence Erlbaum Associates Ltd. ISBN 0863773141

[10] Reich, Shuli, S. and Ruth, Pauline. 1982. Wason's selection task: verification, falsification and matching. British Journal of Psychology, 73, 395-405.

[11] Cosmides, L. (1989). The logic of social exchange: Has natural selection shaped how humans reason ? Studies with Wason's selection task. Cognition. 31, 187-276.

[12] Manktelow, K. I. and Evans, J. St. B. T. (1979). Facilitation of reasoning by realism: Effect or non-effect? Biritish Journal of Psychology, 70, 477-488.

[13] Cox, J. R. and Griggs, R. A. (1982). The effects of experience on performance in Wason's selection task. Memory and Cognition, 10, 496-502.

[14] Reich, S.S. and Ruth, P. (1982). Wason's selection task: verification, falsification and matching. British Journal of Psychology, 73:3, 395-404.

[15] Evans, J. St. B. T and Lynch, J. S. (1973). Matching bias in the selection task. British Journal of Psychology, 64, 391-397.

[16] Hilgard, E. R. 1938. A summary of alternative procedures for the construction of Vincent curves. Psychology Bulletin, 35, 282-297.